PATENT
Docket No.: 200300842-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellant	Osecky, et al.	Examiner	Ahmed, Enam
Serial No.	10/726,976	Group Art No.	2112
Filed	12/03/2003	Confirmation No.	5119
For	FAULT-DETECTING COMPUTER SYSTEM		

July 8, 2008

Mail Stop Appeal Brief -- Patent
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

In accordance with 37 C.F.R. § 41.37, and responsive to the Non-Final Office Action dated 4/08/2008, Appellants hereby file this appeal brief in support of their appeal in the above-identified matter. A notice of appeal, with appropriate fee of $510.00 as required by §§41.31, 41.20(b)(1), is filed herewith. The $510.00 fee for this appeal brief, as required by 37 C.F.R. §41.20(b)(2), is filed herewith. This appeal brief is timely filed within two months of the mailing of the notice of appeal.

I. *Real party in interest*

The real party in interest for this appeal is Hewlett-Packard Development Company, L.P. (HPDC), a limited partnership established under the laws of the State of Texas and having a principal place of business at 20555 S.H. 249, Houston, TX 77070, U.S.A. HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, L.L.C. Evidence of this assignment, which was recorded on December 3, 2002, may be found at reel/frame 014773/0122.

II. *Related appeals and interferences*

No other appeals or interferences are known to appellants, the Appellants' legal representative, or assignee which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

III. *Status of claims*

All pending claims, 1-36, stand twice rejected.

Claims 1-36 are appealed herein.

IV. *Status of amendments*

There are no pending amendments to the present application.

V. *Summary of Claimed Subject Matter*

The present system comprises a method for detecting computational errors in a digital processor executing a program. The method, as recited in claim **1**, comprises:

separating (406) the program into computation (code) segments 207 [Figs. 2, 3, and 4, paragraphs 0018, 0036, and 0052];

compiling (407) source code 201 for at least one of the segments to generate two code sections 303/305, one of which is functionally redundant with respect to the other [Figs. 3 and 4, paragraphs 0036, 0037, and 0053];

generating (407/408) comparison (verification) code 304/308 for comparing results produced by execution of the two code sections 303/305 [Figs. 3 and 4, paragraphs 0037, 0042, and 0054];

executing (420) each of the code sections 303/305 in a different computational domain 206(1)/206(2) to generate respective results [Figs. 3 and 4, paragraphs 0020, 0038, and 0055];

comparing (220/310/430) the respective results using the comparison code 304/308 [Figs. 3 and 4, paragraphs 0026 and 0055]; and

executing one of the code sections to alter further flow of execution of the program only if (430) the respective results are identical [Figs. 3 and 4, paragraphs 0031 and 0055].

Independent claim **18** recites a system comprising a compiler 202 configured to:

separate (406) a program into computation segments [Figs. 3 and 4, paragraphs 0018 and 0036];

compile (407) source code for at least one of the computation segments to generate output comprising two redundant code sections 303/305, each of which is configured to execute in a different computational domain 206(1)/206(2) [Figs. 3 and 4, paragraphs 0020, 0038 and 0053]; and

generate (407/408) comparison code 304/308 for comparing respective results produced by execution of the two code sections [Figs. 3 and 4, paragraphs 0026, 0031 and 0055].

Independent claim **31** is in means-plus-function form, and recites a system comprising:

means for compiling source code (processor 101 [Fig. 1], which corresponding function is performed by compiler 202 [Figs. 2 and 3], as described in paragraphs 0018 and 0019);

means for generating comparison (error-handling) code (processor 101 [Fig. 1], which corresponding function is described in paragraph 0020);

means for comparing (processor 101 [Fig. 1], which corresponding function is described in paragraphs 0026 and 0042); and

means for performing error handling (error recovery) (processor 101 [Fig. 1] which corresponding function is described in paragraphs 0032, 0033, and 0042).

Independent claim **34** is directed to a software product comprising instructions, stored on computer-readable media, wherein the instructions, when executed by a computer, perform steps for detecting computational errors in a digital processor executing a program. Claim 34 includes essentially the elements of claim 1, as described above.

VI. *Grounds of Rejection to Be Reviewed on Appeal*

1. Claims 1–3, 6, 18, 19, 21, 24, 29–32, 34, and 35 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al. in view of U.S. Patent No. 7,043,728 to Galpin.

2. Claims 4, 17, and 25 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 7,168,008 to de Bonet.

3. Claims 5 and 36 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 6,880,112 to Lajolo.

4. Claims 7, 8, 26, 27, and 33 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No.

7,043,728 to Galpin, and further in view of U.S. Patent No. 5,537,559 to Kane et al.

5. Claims 9–14 and 20 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 6,862,608 to Merkey.

6. Claims 15–16 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 7,110,431 to Oates.

7. Claim 22 stands rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No.6,446,058 to Brown.

8. Claim 23 stands rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, U.S. Patent No.6,446,058 to Brown and further in view of U.S. Patent No. 7,110,431 to Oates.

9. Claim 28 stands rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, U.S. Patent No. 5,537,559 to Kane et al., and further in view of U.S. Patent No. 6,862,608 to Merkey.

VII. *Argument*

Summary

Appellants maintain (1) that neither the Bunnel nor the Galpin reference teaches or suggests each of the claim elements / limitations in any of the claims pending in this appeal (claims 1–36), and (2) that the Examiner's suggested motivation for combining the two references is entirely unrelated to the function, structure, or purpose of Appellants' invention as claimed in each of the independent claims. Appellants thus maintain that none of the pending claims are obvious under 35 U.S.C. 103(a) in view of the applied references.

Discussion

1. Claims 1–3, 6, 18, 19, 21, 24, 29–32, 34, and 35

1.1 Independent claims 1, 18, 31, and 34

Independent claims 1, 18, 31, and 34 were rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al. in view of U.S. Patent No. 7,043,728 to Galpin.

In the Office Action dated 04/08/2008, the Examiner stated:

> With respect to claim 1, the Bunnel et. al, reference teaches separating the program into computation segments (column 3, lines 9-22), (column 3, lines 42-47), (column 7, lines 8-40), and (column 8, lines 58-61); compiling source code … to generate two code sections (column 7, lines 8-27); executing each of the sections in a different computational domain … (column 1, lines 28-50); (column 2, lines 30-46) and (column 16, line 65 – line 17, column 10).

1.1.1 Bunnel does not teach Appellants' claimed limitation of separating the program into computation segments.

Appellants note that Bunnel does not describe separating a program into "computation segments". Column 3, lines 9 - 18 of Bunnel state:

> The operating system, upon execution by the processor, provides for the reservation of a first portion of the memory address space for support and application programs, … a second portion for dynamic allocation and recovery by the operating system as necessary for the execution of support and application programs, and **a third portion,** located within said second portion, **for the static storage**, at predefined addresses, **of the executable code segments of the support and application programs** [emphasis supplied].

The "executable code segments" referred to above are not "source code" segments into which a program is separated *prior to compilation*, as recited in Appellants' independent claims, since source code is not executable until after it is compiled. Bunnel neither teaches nor suggests "separating the program into computation segments" prior to compilation of the code, as recited in Appellants' independent claims.

1.1.2 Bunnel does not teach Appellants' claimed limitation of compiling source code ... to generate two functionally redundant code sections.

Column 3, lines 42-47, in the Bunnel reference state:

> Another advantage of the present invention is that it preserves a maximum size of RAM main memory space for transient application programs by **minimizing duplicate instantiations of program code segments**. Execution delay due to repeated copying of program code segments is also generally eliminated [emphasis supplied].

In the above section, Bunnel makes it clear that "duplicate instantiations of program code segments" are to be "minimized". In Appellants' claimed invention, duplicate instantiations of each program segment *must* be present. Therefore, Bunnel cannot possibly be used as a reference which teaches segmentation of a program into "functionally redundant" codes sections, (as recited in Appellants' independent claims), since Bunnel's goal is to *minimize* the possibility of any such duplicate instantiations.

In column 7, at lines 10-16, Bunnel states:

> Program segments typically fall into one of two classes: code and data. **Code segments, typically one per application program**, contain the actual executable code of the application program. A data segment loaded simultaneous with the loading of a code segment permits image establishment of the predefined and preinitialized program data structures [emphasis supplied].

In column 7, at lines 11-12 (above), Bunnel states "code segments, *typically one per application program*, contain the actual executable code of the application program". Thus, Bunnel makes it clear that a given program is not intended to be divided into "functionally redundant" code sections, since there is typically only one code or computation section *per application* in Bunnel's system.

Thus Bunnel does not "compil[e] source code ... to generate two code sections", as claimed by Appellants, since Bunnel clearly states quite clearly that there is typically only one "program segment" per application.

In column 8, lines 58-61, Bunnel states:

> Finally, a stripped code segments area 78 is provided in the nonvolatile portion 62 of the address space 60. This area 78 provides for the execution ready image storage of application program code segments.

The above section in Bunnel teaches nothing related to the existence of "functionally redundant" codes sections, as recited in Appellants' independent claims. Read in the context of the entire reference, the above text merely indicates that area 78 of address space 60, which is a non-executable area (indicated as ROM memory in Figure 3) in which code is stored, is used to store segments of a *single* application program.

The Examiner states that "there are sections [in the Bunnel reference] which discuss compilation or execution of [a] program to generate multiple code sections. Thus the Bunnel reference teaches "compil[e] source code... to generate two code sections" (column 7, lines 8–27).

With respect to the above statement, Appellants again note that column 7, lines 11-12 of Bunnel state "**code segments, typically one per application program**, contain the actual executable code of the application program". Thus, Bunnel makes it clear that a given program, or application, is not divided into multiple "computation segments".

In addition, the Bunnel reference states:

> "... where the size of transient program area is constrained, or where the access response time must be short, **it is undesirable to have to duplicate instances of each program into the**

> **transient program area in order to permit execution**. " [column 9, lines. 5–9, emphasis supplied].

Therefore, since Bunnel's goal is to *minimize* the number of duplicate program code segments, the Bunnel reference cannot be considered relevant to Appellants' claimed invention, in which duplicate instances of each program segment MUST co-exist. Bunnel thus does not teach generating or executing multiple code sections for a particular application, and clearly teaches away from Appellants' fundamental claimed inventive concept which *requires* the execution of duplicate instances of application program segments.

1.1.3 Bunnel does not teach Appellants' claimed limitation of executing each of the sections in a different computational domain.

Appellants can find nothing in either column 1, lines 28-50, or in column 2, lines 30-46, that mentions "executing each of the sections in a different computational domain". In column 16, line 65 – column 17, line 10, Bunnel states, in part:

> If, however, the expand flag was set, a control state 152 is entered instead. Here, the OS boot loader directly allocates the required size of the ".bss" segment without requiring the segment to be actually downloaded."

Appellants suggest that there is nothing in the above section, or in any other section in the Bunnel reference directed to the execution of code sections "in a different computational domain".

Thus, Appellants maintain that Bunnel neither teaches nor suggests any of the claimed elements of "separating the program into computation segments"; "compiling source code ... to generate two code sections"; or "executing each of the sections in a different computational domain", as recited in each of Appellants' independent claims. The Galpin reference is the only reference in addition to Bunnel that was cited in the rejection of Appellants' independent claims. Based on the above arguments, it is irrelevant as to whether the Galpin reference teaches the additional claim elements suggested by the Examiner, since Galpin does not supply the critical elements of Appellants' claim 1 that are not present in the Bunnel reference.

1.1.4 Galpin does not teach Appellants' claimed limitation of generating comparison code for comparing results produced by the execution of the two code sections.

The Examiner states that the Galpin reference teaches "generating comparison code for comparing results produced by the execution of the two code sections (column 2, lines 29-48), (column 2, line 62 - column 3, line 9), (column 3, lines 21-30), and (column 8, lines 28-46)". However, Appellants note that none of the referenced sections in Galpin make any mention whatsoever of "generating comparison code". Galpin teaches "comparing states of the first and second processes". Appellants assert that Galpin does not teach the use of generating comparison code to compare the results of the execution of two processes, but instead, as noted, only compares the states of the processes.

1.1.5 Galpin does not teach Appellants' claimed limitations of comparing the respective results using the comparison code and executing one of the code sections to alter further flow of execution of the program only if the respective results are identical.

The Examiner states that the Galpin reference teaches "comparing the respective results using the comparison code and executing one of the code sections to alter further flow of execution of the program only if the respective results are identical (column 2, lines 29-48), (column 2, line 62 - column 3, line 9), and (column 3, lines 21-30)". However, Appellants note that the system described by Galpin *has already executed a particular code section* by the time the Galpin system has detected a discrepancy in "the event that one of the processes does not complete its respective instruction sequence, or if the process states do not otherwise favorably compare" (column 2, lines 52-55). This is corroborated by the next sentence in the Galpin reference, which states "the method calls for rolling back the processes to performing error diagnostics and/or rolling back the processes to their most recent respective states of agreement, e.g., the states prior to execution of the first and second sequences, and retrying execution of those sequences" (column 2, lines 56-61).

Thus, Galpin does not function in the manner of Appellants' claimed invention, in which *a code section is not executed* to alter further flow of execution of the program *unless* if the respective results are identical, and therefore the Galpin reference does not teach "comparing the respective results using the comparison code and executing one of the code sections to alter further flow of execution of the program *only if* the respective results are identical.

1.1.6 The Examiner has failed to provide a proper rationale for combining the cited references.

With respect to the motivation to combine the references, as required for each of the present rejections, Appellants note the following reasons why the Examiner's suggested motivation to combine the Bunnel and Galpin references is improper.

The Galpin reference states, at column 6, lines 7-12:

> As noted above, devices 10, 12 on both sides of the redundant pair have synchronization, or sync, lines 38 running between them that are used **to keep their operation synchronized**. This **enforces loose coupling** between the devices 10, 12 and, more particularly, between their respective processing sections 18, 20 [emphasis supplied].

Appellants thus maintain that the synchronization required by Galpin negates the applicability of the reference to Appellants' claimed invention, in which there is *no* requirement for synchronization of the redundant segments, either claimed, or indicated in the Specification.

Appellants note that a fundamental aspect of their claimed invention is ensuring that the redundant processes are either run at different times or in different address spaces to ensure that disruptive events do not affect both code segments in the same manner. This fundamental aspect is contrary to the teaching of Galpin, which thus teaches away from Appellants' claimed invention. This 'teaching away' is a 'secondary consideration' of nonobviousness that must be taken into account in assessing the applicability of the cited art as well as the nonobviousness of the claimed invention.

The Examiner states that the motivation to combine the Bunnel and Galpin references "can be provided such as for control and more particularly, for example, for fault-tolerance and fault-detection in process control". However, the Examiner's stated motivation provides no explanation of why anyone of ordinary skill in the art would consider combining the Galpin reference with a reference (namely, Bunnel) that is directed to "*operating system architecture* suitable for execution from ROM directly or down-loadable through a communications connection" in an attempt to yield Appellants' claimed system, which is not related to operating systems or "operating system architecture". This is particularly true given the fact that the combination of references fails to teach or suggest a number of the elements of Appellants' claimed invention.

In addition, the Examiner has failed to show how the two systems described in the two references could be combined to yield an operational system at all, particularly since any attempted integration of the systems of Bunnel and Galpin would result in a conflict between Bunnel's '*one* code segment per application program' and Galpin's requirement for *two* code sections per application. Two references cannot be properly combined where the resulting combination would destroy the inherent or intended function of one of the references. Furthermore, MPEP sec. 706.02(j) [Contents of a 35 U.S.C. 103 Rejection] states in part:

> ...the examiner should set forth in the Office action ... (C) the proposed modification of the applied reference(s) necessary to arrive at the claimed subject matter ...

In the present case, however, the Examiner has not described how either of the references could actually be modified to effect the proposed combined system. The Examiner is obligated to provide some *valid* justification to combine / modify the references - either from the references themselves or from some clearly identified, well known principle in the art. "Rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." [see Examination Guidelines for Determining Obviousness Under 35 U.S.C. 103 in View of the Supreme Court Decision in *KSR International Co.* v. *Teleflex Inc,* Federal Register, Vol. 72, No. 195, p. 57528, under sec. III. Rationales To Support Rejections Under 35 U.S.C. 103].

Furthermore, "[t]he rationale to support a conclusion that the claim would have been obvious is that the substitution of one known element for another would have yielded predictable results to one of ordinary skill in the art at the time of the invention" [57530, *supra*, B. Simple Substitution of One Known Element for Another To Obtain Predictable Results, item (4)]. However, in the instant case, the Examiner fails to explain how "predictable results" can be achieved, with respect to yielding Appellants' claimed invention, by combining elements, picked from unrelated sources, that do not even collectively yield the claimed invention.

It thus remains the Examiner's burden to devise an alternative rationale -- *why* would a person of ordinary skill randomly select such a collection of parts from unrelated references? The Examiner has failed to meet this burden, and thus, for this reason alone, Appellants' claimed invention has not been shown to be obvious in view of the applied references.

1.1.7 The combination of Bunnel and Galpin fails to teach or suggest each of the elements in Appellants' independent claims.

The Bunnel reference, as explained above, does not teach or suggest any of the claimed elements of "separating the program into computation segments"; "compiling source code … to generate two code sections"; or "executing each of the sections in a different computational domain", as recited in each of Appellants' independent claims.

Furthermore, as explained above, the Galpin reference neither teaches nor suggests "generating comparison code for comparing results produced by the execution of the two code sections" or "executing one of the code sections to alter further flow of execution of the program only if the respective results are identical".

The combination of Bunnel and Galpin is not only functionally incompatible with both the general and specific idea of Appellant's system and method, but also fails to teach or suggest each of the elements in Appellants' independent claims.

1.2 Dependent claims 2, 3, 6, 19, 21, 24, 29, 30, 32, and 35

Appellants believe that an analysis and rebuttal of the Examiner's reasons for rejecting Appellants' dependent claims is unnecessary, as those reasons are moot in view of the evidence and reasoning, presented above, which shows that all of the pending independent claims,1, 18, 31, and 34 should be allowable over the cited art.

Appellants believe that all of the pending dependent claims are also allowable, since each of these claims depends from a claim which Appellants believe has been shown to be allowable.

1.2.1 Claims 2 and 35

The Examiner states that "… Bunnel et. al. teaches wherein said computational domain comprises a time domain (column 9, lines 5-12), (column 1, lines 29-51), (column 4, lines 26-40), and (column 4, line 49 - column 5, line 16)".

Bunnel, at column 9, lines 5-12, states:

> However, where the size of transient program area is constrained, or where the access response time must be short, it is undesirable to have to duplicate instances of each program into the transient program area in order to permit execution. Thus, the present invention provides the ability to maintain the code segments of programs not in a file system format, but rather in the stripped code segments area 78 in a format that may be directly executed.

The above paragraph provides no indication whatsoever of a computational domain comprising a *time* domain. It simply addresses program formatting.

Appellants also submit that there is nothing in the other cited sections of Bunnel that describes anything remotely similar to or relevant to a "time domain". Thus claims 2 and 35 cannot be rendered obvious by these unrelated teachings.

1.2.2 Claims 3, 24, and 32

The Examiner states that "...Bunnel et. al. teaches wherein compiling the source code to schedule execution thereof so that a minimum number of processor clock cycles elapse between execution of a first one of the code sections and execution of the other one of the code sections (column 3, lines 3-18), (column 3, lines 42-47) and (column 8, lines 31-36)".

Appellants point out that neither the term "clock cycle" nor even the word "clock" appears anywhere in the Bunnel reference, nor is any related concept mentioned. Thus claims 3, 24, and 32 cannot be rendered obvious by the unrelated teaching.

1.2.3 Claim 6

The Examiner states that "...Bunnel et. al. teaches wherein the compiling step includes compiling the source code such that each of the code sections is executed using separate resources of the processor (column 3, lines 3-22) and (column 14, lines 20-34)".

In column 3, lines 3-22, Bunnel states, in part:

> ...a third portion, located within said second portion, for the static storage, at predefined addresses, of the executable code segments of the support and application programs.

Thus Bunnel compiles *all* source code into a *single* portion of memory.

In column 14, lines 20-34, Bunnel states, in part:

> ...If this operating system kernel is to be downloaded to a computer system 10, then the uninitialized portion of the data segment, specifically the ".bss" segment, is preferably deleted from the kernel image....

Here, Bunnel addresses loading an operating system kernel, and fails to mention anything related to the subject of executing redundant code sections using separate resources of a processor. Thus, for at least the above reasons, claim 6 cannot be rendered obvious by the unrelated teachings.

1.2.4 Claims 19 and 21:

Claim 19 was rejected for essentially the same reason as claim 2. Applicants thus submit the same argument as provided above with respect to claim 2, and assert, for the same reasons set forth therein, that claim 19 is patentable over the cited references.

Claim 21 stands or falls with claim 19, from which it depends.

1.2.5 Claim 29

With respect to claim 29, the Examiner states that "the Galpin references teaches wherein the compiler uses an explicit scheduling aspect of the processor's instruction set to ensure that the two code sections are each executed by a different set of functional units ...".

Appellant notes that the cited sections in Galpin discuss "sender" and "listener" processors that "synchronously step through sequential schedules". This reference is thus entirely off-point with respect to Appellants' claim 29, since Appellants' claimed system operates *asynchronously.* Thus claim 29 cannot be rendered obvious by the unrelated teaching.

1.2.6 Claim 30

Appellants believe that claim 30 is allowable over the cited art for the sames reasons set forth in sections 1.1.3 – 1.1.5, above.

2. Claims 4, 17 and 25

Claims 4, 17, and 25 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 7,168,008 to de Bonet.

2.1 Claims 4 and 25

The Examiner states that "The de Bonet reference teaches wherein said minimum number of processor clock cycles is predetermined as a function of statistical properties of duration of disruptive events causing said computational errors (column 7, lines 33-42)."

de Bonet states, at column 7, lines 33-42:

> Faults caused by overlong execution time of a protected software component 320 may also be detected by embodiments of the present invention. These types of faults are typically caused by code within the protected software component 320 which places the software system into an infinite loop, or causes portions of code to execute for an unusually long time. Certain embodiments of the present invention detect a fault of this type by comparing the time taken to execute protected software component 320 with a predetermined time period.

The above section in the de Bonet reference discusses program execution 'hanging' in either an infinite loop or executing "for an unusually long time". This paragraph bears no relation whatever to predetermining a minimum number of processor clock cycles as a function of *statistical properties of the duration of a disruptive event.* Thus claims 4 and 25 cannot be rendered obvious by the unrelated teaching.

2.2 Claim 17:

With respect to claim 17, the Examiner states that "the de Bonet reference teaches wherein the step of compiling the source code is performed by incrementally translating the source code (column 11, lines 47-67)".

Above, the examiner cites claim 13 in the de Bonet reference, which states:

> A data processing system readable medium, comprising code containing instructions translatable for: designating a first software component of a computer program as protected; wherein the first software component comprises a series of instructions to be executed on a central processing unit (CPU); saving a state of the computer program before executing the first software component; associating a remedial software component with the first software component; executing the first software component; determining whether a fault occurred during execution of the first software component, wherein the fault is caused by one or more programming defects within the first software component; and if the fault is detected, terminating the execution of the first software component; restoring the computer program to the previously saved state; and executing the remedial software component.

Appellants note that the "translatable" instructions referred to above are not related to the step of compiling the source code by incrementally translating the source code, as recited in Appellants' claim 17. Thus claim 17 cannot be rendered obvious by the unrelated teaching.

3. Claims 5 and 36

Claims 5 and 36 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 6,880,112 to Lajolo.

The Examiner states that "The Lajolo reference teaches wherein said computational domain comprises a spatial domain (column 3, lines 25-48)".

Column 3, lines 25-48, of the Lajolo reference state, in part:

> ...One of the key issues when using an electromagnetic simulator is the gridding process or, in other terms, how the structure under analysis is discretized and decomposed in spatial basis functions. The goal was to have a convergence of the simulation process on a 1 .mu.m width...

In view of the above, Appellants assert that there is nothing in column 3, lines 25-48 (or in anywhere else) of the Lajolo reference that addresses the subject of computational domains. Lajolo is concerned with "how [a] structure under analysis is discretized". Thus claims 5 and 36 cannot be rendered obvious by the unrelated teaching.

4. Claims 7, 8, 26, 27, and 33

Claims 7, 8, 26, 27, and 33 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 5,537,559 to Kane et al.

The Examiner states that "the Kane et. al. reference teaches wherein said resources comprise functional units and partitioned registers", and adds that the Kane reference provides "the motivation for wherein said resources comprise functional units and partitioned registers is for parallel execution of instructions or maximum throughput".

Appellants assert that the Kane reference has no significant relationship (if any at all) to Appellant's claimed invention, or to claims 7, 8, 26, 27, and 33, which recite utilizing functional units and partitioned registers for the purpose of error detection, which purpose and related function is not related in any manner to "parallel execution of instructions or maximum throughput". If the Examiner's suggested motivation is either the teaching or the motivation supplied by the Kane reference, then, Appellants assert that the system resulting from the cited combination of references cannot render claims 7, 8, 26, 27, and 33 obvious, given the unrelated teaching and non-relevant 'motivation' to combine.

5. Claims 9–14, and 20

Claims 9–14 and 20 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 6,862,608 to Merkey.

Appellants allow that claims 12 and 13 stand or fall together with respective independent claim 1.

With respect to claim 9, the Examiner cites the Merkey reference and then states that "the motivation for wherein the partitioned registers are utilized by encoding register names from a first set of registers into instructions in a first one of the code sections and encoding register names from a second set of registers into instructions in the other one of the code sections is to improve system performance".

Appellants note that the Merkey reference is concerned with partitioning *disk* storage devices, and fails to even mention partitioned registers. Thus claim 9 cannot be rendered obvious by the unrelated teaching.

With respect to claims 10 and 20, the Examiner cites the Merkey reference as teaching "wherein the respective results are compared by executing the comparison code in a different computational domain from the domain in which one of the code sections was executed (column 12, lines 42-47)".

The cited section of Merkey discusses 'registering' device objects in Linux, and does not mention anything remotely related to "executing comparison code". Thus claims 10 and 20 cannot be rendered obvious by the unrelated teaching.

With respect to claim 11, the Examiner states that "the Galpin references teaches wherein the compiler uses an explicit scheduling aspect of the processor's instruction set to ensure that the two code sections are each executed by a different set of functional units ...".

Appellant notes that the cited sections in Galpin discuss "sender" and "listener" processors that “synchronously step through sequential schedules”. This reference is thus entirely off-point with respect to Appellants’ claim 11, since Appellants’ claimed system operates *asynchronously.* Thus claim 11 cannot be rendered obvious by the unrelated teaching.

With respect to claim 14, the Examiner states that the Bunnel reference “teaches wherein each of the computation segments receives a set of inputs, performs at least one computation on the input values, and exposes a set of outputs to further computation (column 1, lines 29-48).

Appellants note, however, that the cited section in Bunnel makes no mention whatsoever of “computation segments”, but rather discusses “embedded controllers”, and notes that “a rather sophisticated operating system is required to simultaneously support a variety of interprocess communication software mechanisms.” Thus claim 14 cannot be rendered obvious by the unrelated teaching.

6. Claims 15 and 16

Claims 15–16 stand rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No. 7,110,431 to Oates.

With respect to claim 15, the Examiner states that “the Oates reference teaches the step of optimizing one of the two code sections to execute via different registers and functional units than the other one of the code sections (column 8, line 62 - column 9, line 11)”.

Column 8, line 62 - column 9, line 11, of the Oates reference state, in part:

> In other aspects, the invention provides an apparatus for efficiently computing a .GAMMA.-matrix as described above, e.g., in hardware. The system includes two registers, one associated with each of l.sup.th and k.sup.th users. The registers hold elements of the short code sequences associated with the respective user such that alignment of the short code sequence loaded in one register can be shifted relative to that of the other register by m elements.

Associated with each of the foregoing registers is one additional register storing mask sequences....

Appellants maintain that this section of the Oates reference deals with the subject of "efficiently computing a .GAMMA.-matrix", and thus bears no relationship to Appellants' claim 15, which recites "optimizing one of the two code sections to execute via different registers and functional units than the other one of the code sections". Thus claim 15 cannot be rendered obvious by the unrelated teaching.

With respect to claim 16, the Examiner states that "the Bunnel reference teaches wherein the compiling step employs code reorganization to dynamically translate the source code into the two code sections" (column 6, lines 10-20) and (column 8, lines 28-40)".

In the above-cited sections, Bunnel respectively discusses "in general, the remaining RAM memory, generally referenced by the reference numerals 50, 50' is utilized as the transient program execution area" and "memory transfer requests directed to the logical interface corresponding to the pseudo-disk device driver results in a transfer of data fully within the main memory 14, though logically consistent with a disk drive paradigm". Neither of these sections is related to using "code reorganization to dynamically translate the source code into [the] two code sections", as recited in Appellants' claim 16. Thus claim 16 cannot be rendered obvious by the unrelated teaching.

7. Claim 22

Claim 22 stands rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, and further in view of U.S. Patent No.6,446,058 to Brown.

With respect to claim 22, the Examiner states "the Brown reference teaches an optimizer for modifying the output of the compiler (column 4, lines 7-27)".

Column 4, lines 7-27 of the Brown reference state, in part:

> ...In particular, output element 157 outputs a match to a predictor 158 for providing a confidence level and the output match to an optimizer 159 for optimizing a solution from the input match. The inference engine 155 is coupled to a knowledge base 156 for receiving created frames and managing frames and slots. The knowledge base accepts input from input/output interface 151 and outputs historical data to output 157...

Appellants maintain, in view of the above section, that Brown does not teach the use of "an optimizer for modifying the output of the compiler *to schedule execution of [the] redundant code.* Since Brown's purpose and function are entirely different than that claimed by Appellants in claim 22, Appellants thus submit that claim 22 cannot be rendered obvious by the unrelated teaching.

8. Claim 23

Claim 23 stands rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, U.S. Patent No.6,446,058 to Brown and further in view of U.S. Patent No. 7,110,431 to Oates.

The examiner states that the "motivation for an optimizer for configuring is for parallel execution of instructions or maximum throughput", and that the "motivation for one of the redundant code sections to execute via different registers and functional units than the other one of the code sections is to manage faults for high availability".

However, Appellants note that their claimed system (including claim 23) has absolutely nothing to do with achieving "maximum throughput" or with "manag[ing] faults for high availability". Appellants' claimed optimizer functions to ensure that, among other things, each of the redundant code sections execute via different registers and functional units so that an error effected by a particularly register set does not affect the results of execution of both code sections.

Appellants assert that the Examiner's asserted statements of motivation to employ the Brown and Oates reference so significantly mis-characterize their claimed invention as to indicate that the cited combination of references must be inapplicable to show obviousness of Appellants' claim 23.

9. Claim 28

Claim 28 stands rejected under 35 U.S.C. 103(a) over U.S. Patent No. 5,594,903 to Bunnel et al., U.S. Patent No. 7,043,728 to Galpin, U.S. Patent No. 5,537,559 to Kane et al., and further in view of U.S. Patent No. 6,862,608 to Merkey.

With respect to claim 28, the Examiner cites the Merkey reference and then states that "the motivation for wherein the partitioned registers are utilized by encoding register names from a first set of registers into instructions in a first one of the code sections and encoding register names from a second set of registers into instructions in the other one of the code sections is to improve system performance".

Appellants note that the Merkey reference is concerned with partitioning *disk* storage devices, and fails to even mention partitioned registers. Thus claim 28 cannot be rendered obvious by the unrelated teaching.

10. Summary

Appellants note that, in order to combine references, there must be at least some relationship between the structure or function of the applied reference and the manner in which the claimed system actually operates. In each case noted above, Appellants assert that the stated motivation to combine the cited references cannot meet this minimum requirement, because each motivation to combine, as suggested by the Examiner, indicates that the applied reference operates in a significantly and patentably different manner than that in which Appellants' claimed invention functions, as described in their Specification.

This significant lack of relevance between Appellants' claimed system and the system which would result from applying the motivation suggested by the Examiner is further indicative of the lack of applicability of the applied references, and further distinguishes Appellants' claimed invention over the references.

In each of the rejections set forth in the present Office Action, Appellants believe that the combination suggested by the Examiner is not similar to Appellants' claimed invention, either in structure, or in function, and therefore such a combination cannot be used to show obviousness of either of these claims or of the claims depending therefrom.

For at least the reasons enumerated above, Appellants believe that independent claims 1, 18, 31, and 34 are allowable over the cited art. Since dependent claims 2-17, 19-30, 32, 33, 35, and 36 incorporate the limitations of the respective independent claims, these claims should also be allowable.

VIII. Claims Appendix.

Appellants enclose a copy of the claims involved in this appeal as an appendix hereto.

IX. Evidence Appendix.

No evidence of previous record is entered or relied upon in this appeal. However, a copy of the authorities relied upon in support of Appellants' arguments presented herein is supplied in the Evidence Appendix.

X. Related Proceedings Appendix.

To Appellants' knowledge, there are no related decisions rendered by a court or the Board for submission with this appeal.

XI. Conclusion

For at least the reasons set forth above, Appellants believe that none of the pending claims 12–21 are obvious in view of the applied references, and thus request that all pending claims be allowed in view of the above discussion.

Other than the costs for the appeal brief, we believe no additional fees are due in connection with this matter. However, if any additional fee is deemed necessary in connection with this brief, the Commissioner is hereby authorized to charge such fee to Deposit Account No. 12-0600.

Respectfully submitted,

LATHROP & GAGE LC

E. Michael Byorick

By____________________________

E. Michael Byorick, Reg. No. 34,131
LATHROP & GAGE L.C.
4845 Pearl East Circle, Suite 300
Boulder, Colorado 80301
Telephone: (720) 931-3000
Facsimile: (720) 931-3001

CLAIMS APPENDIX

1. A method for detecting computational errors in a digital processor executing a program, the method comprising steps of:

separating the program into computation segments;

compiling source code for at least one of the segments to generate two code sections, one of which is functionally redundant with respect to the other;

generating comparison code for comparing results produced by execution of the two code sections;

executing each of the code sections in a different computational domain to generate respective results;

comparing the respective results using the comparison code; and

executing one of the code sections to alter further flow of execution of the program only if the respective results are identical.

2. The method of claim 1, wherein said computational domain comprises a time domain.

3. The method of claim 1, wherein the compiling step includes compiling the source code to schedule execution thereof so that a minimum number of processor clock cycles elapse between execution of a first one of the code sections and execution of the other one of the code sections.

4. The method of claim 3, wherein said minimum number of processor clock cycles is predetermined as a function of statistical properties of duration of disruptive events causing said computational errors.

5. The method of claim 1, wherein said computational domain comprises a spatial domain.

6. The method of claim 1, wherein the compiling step includes compiling the source code such that each of the code sections is executed using separate resources of the processor.

7. The method of claim 6, wherein said resources comprise functional units and partitioned registers.

8. The method of claim 7, wherein the partitioned registers are used to effect the detection and repair of errors in the registers and paths to/from the registers.

9. The method of claim 6, wherein the partitioned registers are utilized by encoding register names from a first set of registers into instructions in a first one of the code sections and encoding register names from a second set of registers into instructions in the other one of the code sections.

10. The method of claim 1, wherein the respective results are compared by executing the comparison code in a different computational domain from the domain in which one of the code sections was executed.

11. The method of claim 1, wherein the compiler uses an explicit scheduling aspect of the processor's instruction set to ensure that the two code sections are each executed by a different set of functional units.

12. The method of claim 1, including performing error handling if a discrepancy between the respective results is found.

13. The method of claim 12, wherein said error handling includes at least one function selected from the group consisting of re-execution, failing, and trapping to an error handling routine.

14. The method of claim 1, wherein each of the computation segments receives a set of inputs, performs at least one computation on the input values, and exposes a set of outputs to further computation.

15. The method of claim 1, including the step of optimizing one of the two code sections to execute via different registers and functional units than the other one of the code sections.

16. The method of claim 1, wherein the compiling step employs code reorganization to dynamically translate the source code into the two code sections.

17. The method of claim 1, wherein the step of compiling the source code is performed by incrementally translating the source code.

18. A system for detecting computational errors in a digital processor executing a program, the system comprising a compiler configured to:

separate the program into computation segments;

compile source code for at least one of the computation segments to generate output comprising two redundant code sections, each of which is configured to execute in a different computational domain; and

generate comparison code for comparing respective results produced by execution of the two code sections.

19. The system of claim 18, wherein the processor:

executes each of the code sections in a different computational domain to generate respective results for each of the code sections;

compares the respective results using the comparison code; and performs error handling, if a discrepancy between the respective results is found.

20. The system of claim 19, wherein the respective results are compared by executing the comparison code in a different computational domain from the domain in which one of the code sections was executed.

21. The system of claim 19, wherein the error handling includes at least one function selected from the group consisting of re-execution, failing, and trapping to an error handling routine.

22. The system of claim 18, including an optimizer for modifying the output of the compiler to schedule execution of the redundant code sections so that a minimum number of clock cycles elapse between execution of a first one of the sections and execution of the other one of the code sections.

23. The system of claim 18, including an optimizer for configuring one of the redundant code sections to execute via different registers and functional units than the other one of the code sections.

24. The system of claim 18, wherein the source code is compiled to schedule execution of the redundant code sections so that a minimum number of processor clock cycles elapse between execution of a first one of the code sections and execution of the other one of the code sections.

25. The system of claim 24, wherein said minimum number of processor clock cycles is predetermined as a function of statistical properties of the duration of disruptive events causing said computational errors.

26. The system of claim 18, wherein the compiler compiles the source code such that each of the code sections is executed using a different set of functional units and partitioned registers of the processor.

27. The system of claim 26, wherein the partitioned registers are used to effect the detection and repair of errors in the registers and paths to/from the registers.

28. The system of claim 26, wherein the partitioned registers are utilized by encoding register names from a first set of registers into instructions in a first one of the code sections and encoding register names from a second set of registers into instructions in the other one of the code sections.

29. The system of claim 18, wherein the compiler uses an explicit scheduling aspect of the processor's instruction set to ensure that the two code sections are not executed by the same functional units.

30. The system of claim 18, wherein the processor:

executes each of the code sections in a different computational domain to generate respective results for each of the code sections;

compares the respective results using the comparison code; and

executes one of the code sections to alter further flow of execution of the program only if the respective results are identical.

31. A system for detecting computational errors in a digital processor executing a program, the system comprising:

means for compiling source code for at least part of the program to generate two code sections, one of which is functionally redundant with respect to the other;

means for generating comparison code for comparing results produced by execution of the two code sections;

wherein each of the code sections is executed in a different computational domain to generate respective results;

means for comparing the respective results using the comparison code; and

means for performing error handling, if a discrepancy between the respective results is found.

32. The system of claim 31, wherein the source code is compiled to schedule execution thereof so that a minimum number of processor clock cycles elapse between execution of a first one of the code sections and execution of the other one of the code sections.

33. The system of claim 31, wherein the source code is compiled such that each of the code sections is executed using a different set of functional units and partitioned registers of the processor.

34. A software product comprising instructions, stored on computer-readable media, wherein the instructions, when executed by a computer, perform steps for detecting computational errors in a digital processor executing a program, comprising:

compiling source code for at least part of the program to generate two code sections, one of which is functionally redundant with respect to the other;

generating comparison code for comparing results produced by execution of the two code sections;

executing each of the code sections in a different computational domain to generate respective results;

comparing the respective results using the comparison code; and performing error handling, if a discrepancy between the respective results is found.

35. The software product of claim 34, wherein said computational domain comprises a time domain.

36. The software product of claim 34, wherein said computational domain comprises a spatial domain.

EVIDENCE APPENDIX

No evidence was submitted into the record by the Examiner which was relied upon by Appellants.

RELATED PROCEEDINGS APPENDIX

To Appellants' knowledge, there are no related decisions rendered by a court or the Board for submission with this appeal.